System Design for the Remote Execution of Library Routines

B. TEUFEL

Department of Computer Science, Swiss Federal Institute of Technology (ETH-Zürich), CH-8092 Zürich, Switzerland

In general the users of personal computers are unable to call library routines – as available on mainframes – from their application programs. Therefore it is necessary to integrate the personal computers into a heterogeneous network with mainframes and to make communication software available to the users of personal computers. This software has to provide a transparent interface, thus allowing conveniently the remote execution of library routines. The subject of this paper is how to design such software for a distributed system.

Received September 1985

1. INTRODUCTION

The successful use of personal computers in a scientific environment depends very much on the available software and the peripheral interfaces of the system. For most of the personal computers on the market today there are application programs available which allow communication with peripheral devices or which support memory and file management. In addition many comfortable program packages, e.g. document editors, are provided.

Unfortunately no possibility exists for the users of personal computers to access subroutines from program libraries like IMSL[1] or Linpack[2] libraries which are well tested and highly reliable. There are of course PC versions of the most popular libraries available. But these PC versions are always reduced versions providing only a part of the breadth of function. The described PC environment is not sufficient – even though it is comfortable in some areas – for those users, who exploit the functionality of program libraries on mainframes. These users should be able to use on the one hand the comfort of a modern PC and on the other hand the customary possibilities of the mainframe. This requires:

– the integration of the PC into a heterogenous computer network;

– the disposal of network software.

In particular, every user whose PC is integrated into a heterogeneous network must have the possibility of accessing program libraries residing on other computers of the network, thus being able to work in a real distributed environment, i.e. to execute a program on different computers and so in different address spaces. This presumes a highly transparent interface, which must offer the same calling syntax for a remote library subroutine as for a local procedure call. This means that most of all the communication and transfer problems must be hidden from the user.

2. MOTIVATION AND BASIC REQUIREMENTS

What possibilities has the user for the subsequent treatment of data calculated on his PC? There are three:

– if the user has a reduced version of a program library available on his computer, he will probably find the right program;

– if he has no such library or if he cannot find the desired program, perhaps he will program the function needed;

– if the user has access to a file transfer program connecting his PC to a mainframe on which the desired program library is available, he can transfer the data for processing on the mainframe.

The last of the three possibilities is not as easy as it seems, because the user has not only to transfer the data from the PC to the mainframe; in general he has to convert the format of his data; then he must have a program for the subsequent treatment of the data on the mainframe; finally he must reconvert the results before transferring back to the PC. Obviously this is not a trivial task.

Therefore the user should be able to call from his application programs on the PC library routines which exist and will be executed on a mainframe. It is necessary for him to perform such a call in a transparent way. This means that the call must be based on a task, doing the data conversion, the data transfer, and most of all the control of communication with the mainframe.

A system allowing the remote execution of library routines has to fulfil two general requirements:

– first, it must be general in the concepts used, so allowing easy change of system environment;

– secondly, it must be special in the given system environment.

Of course these two requirements seem to be contradictory. The designer of such a system has to find a way to combine these two opposed statements, as will be described later.

For the following description it is assumed that the user of the PC can connect his PC to a computer network, for example in the simplest way via an RS 232 interface. Hence it is further assumed that he can communicate with the network and the mainframe by using ASCI character sequences. Then a system which allows the remote calls of library routines may be seen as an application program, belonging to the seventh layer – the application layer – of the ISO Reference Model for Open Systems Interconnection (OSI).[3,4]

3. STRUCTURES OF THE COMMUNICATION SERVERS

By calling a procedure associated with another computer, it is always necessary to open a session between the two computers involved using a communication medium. Then the parameters must be transferred. After doing this the called computer and then the server running on it are able to call and execute the desired procedure. The

called computer informs the one calling when the execution of the procedure has finished in order to prepare the transfer of the results.[5] On the local machine, i.e. on the calling PC, the program execution then has to be continued in the same way as it will be continued after a single-machine procedure call.[6] This principle will be found on all procedure calls, involving more than one computer. It shows that the remote call and execution of library routines is a synchronous program-driven transfer of control between programs working in different address spaces.

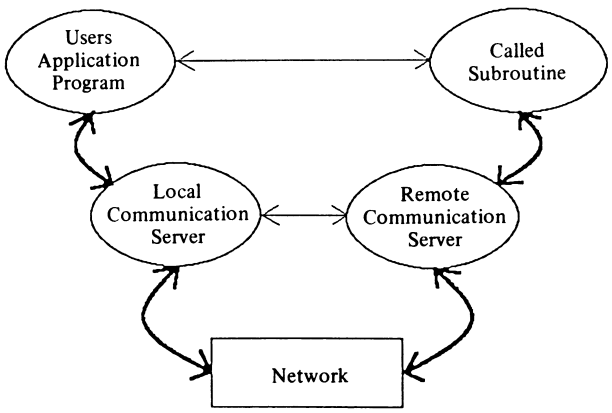

Fig. 1. Communication and data flow paths for the remote execution of procedures. —, Communication path; —, data flow path.

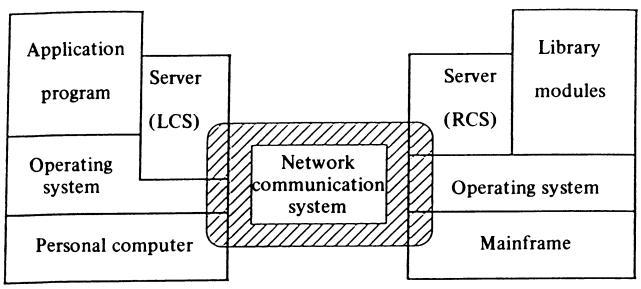

Fig. 2. System structure of ICARUS.

As discussed by B. J. Nelson, there are five crucial issues in the mechanisms of remote procedure calls: uniform call semantics, binding and configuration, strong typechecking, parameter functionality, and concurrency and exception control.[7] These issues are elaborated and a set of properties for remote procedure call mechanisms are defined. Although this was performed for mechanisms which are fully and uniformly integrated into a programming language for a homogeneous distributed system, Nelson's remarks are useful for the design of our system (a system which is not integrated into a programming language and which has to work in a heterogeneous environment).

It is not trivial to structure distributed programs and there have been many proposals to do this.[8] In Fig. 1 the basic communication and data flow paths are shown, which were used for the remote execution of library routines in a distributed computer system. The user only knows the communication path between his application program and the called library subroutine. Parameters which he wants to transfer to a library routine go 'physically' at first to a so-called Local Communication Server. This Local Communication Server (LCS) transfers the parameters via the network to the Remote Communication Server (RCS). Finally this server calls the subroutine and transfers the parameters to the called subroutine. The embedding of these servers into a real system is shown in Fig. 2 describing the system structure of the prototype system ICARUS.

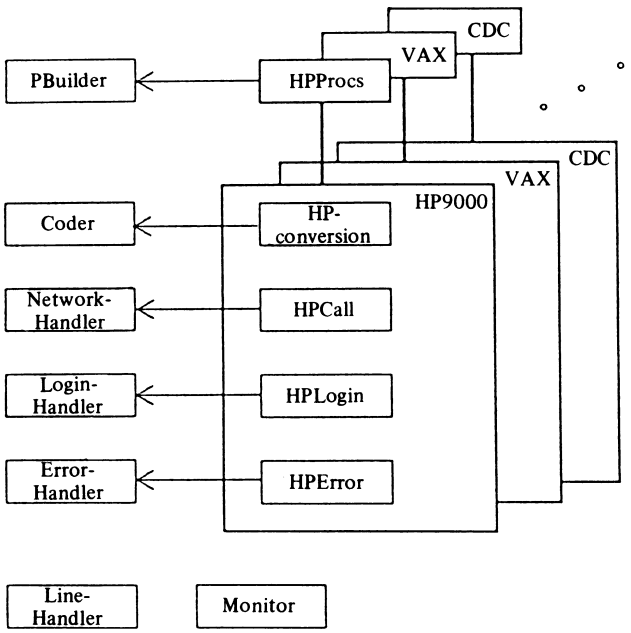

Fig. 3. Local Communication Server.

Both communication servers must have a modular design. This is specially indispensable for the LCS, for being able to expand the system to new hosts. Fig. 3 shows that the Line-Handler of the LCS can be designed independently. But the Coder, the Network-, Login-, and Error-Handler have to import host-specific procedures. The modularisation of the RCS in a UNIX environment is shown in Fig. 4. Both concepts are realised with ICARUS running on the Lilith personal computer[9] and the Hewlett-Packard HP 9000 as the mainframe. A detailed description of ICARUS has been reported in Ref 10. As mentioned above, the syntax of a call to a procedure with the object code residing on another computer should be the same as for local procedure calls. This could either be done by integrating the communication process into a programming language or by reducing the problem of the remote execution of arbitrary routines to the remote execution of a well-known set of library routines. If additionally a programming language is used which allows good modularisation, e.g. by providing import and export structures as available in MODULA-2,[11] it will be possible to offer a calling syntax which does not differ from local procedure calls.

The necessary data transfer must be controlled by a protocol. Because general concepts are to be used it must be recognised that the data transfer occurs physically over lines to terminals. This means that the terminal as well as the underlying operating system interprets some control sequences. Therefore we can transfer binary data only when it is coded into printable ASCII characters.

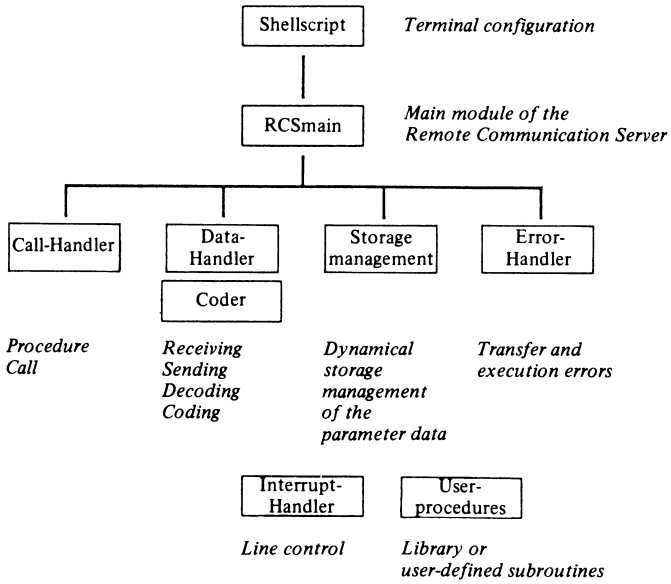

Fig. 4. Remote Communication Server.

Now the transfer protocol is responsible for building data packets and for calculating a checksum. Moreover, there must be a handshake mechanism for the correct communication.

4. PROCESS SYNCHRONIZATION

The establishment of a session between the involved computers should be done with simple communication mechanism. Such mechanisms can be divided into synchronous and asynchronous ones.[12] For the definition of a system allowing remote subroutine execution it will be advantageous to use synchronous communication mechanisms. This gives the system designer better possibilities for error detection, especially in the case when one of the involved computers is shut down during an open session.

To establish a session between two computers three steps have to be executed:

- configuration of the network interface;
- addressing the host through the network address;
- logging in at the host.

For the configuration of the network interface one has to analyse and if necessary store the present status of the interface module. Analysing the interface means setting the parameters of the network in a useful way. All this, as well as the addressing of the host, can be done with special network commands that should be provided by the network. In addition the network should send error messages in the case of incorrect situations. But the designer of such a distributed system should know that situations may occur where the network does not respond to any action. These situations can only be detected using timeout variables. Obviously the setting of these timeout variables depends on the system environment.

More complex and dependent on the host is the third step: the login. Unfortunately there is no standardisation for the login procedures of different computers even when run on the same operating system, e.g. UNIX or VMS. Therefore the system designer has to analyse the login structure of each computer he wants to integrate into his distributed system. Additional to this there exist numerous exceptional situations.[13] Some of these exceptional situations can again be detected only by using timeout variables. The system designer should know that the more he is forced to use timeout variables, the greater the probability that the system performance will be reduced.

When all the basic operations have been performed the essential communication with the host can be started. After the successful login we have to set the configuration of the terminal. Here one of the arguments for the second basic requirement postulated in Section 2 appears. From the viewpoint of the program system on the PC each integrated host is unique. All hosts must have only a common interface to the PC program system. But this interface can be realised on each host in a very system-dependent way. Especially when the host runs for example under a UNIX operating system, there will be many tools available which allow a convenient definition of the interface.

After the configuration of the terminal we have to start the RCS, providing a controlled communication base. Depending on the workload of the host, the terminal configuration and the start of the RCS take different time intervals at different moments. Therefore the RCS sends a synchronising signal to the waiting LCS. A waiting LCS does not imply that the PC must wait until the synchronisation is done. One can assume that it is also possible that other calculations can be done in parallel. But this depends highly on the operating system (multi-tasking) and the hardware interface (the question whether the physical interface generates interrupts or not). Obviously it is not only the time interval till synchronisation that can be used on the PC for further calculations, but also the time interval when the host is executing the desired library routine. This will lead to real distributed and parallel computing.

Receiving the synchronising signal, the LCS knows that the RCS is ready to receive the parameter data. Therefore the local server starts the transfer and both servers are synchronised after each transferred data packet. This allows a very quick detection of system failures.

5. ERRORS

As mentioned above, there are many situations producing errors. Thus the user must be able to have access to status variables where he can detect whether an action is correct or not. But in the case of an error at first the system has to try to cancel the error. This can be done for example if a checksum error occurred by the data transfer. Only if there is no success for the system in cancelling the error, a status variable must be set. The user of the system has then to decide what should be done.

There can arise numerous exceptional situations while transferring the data or while the host is executing the library subroutine. Hard errors, i.e. errors by which the two communication servers can no longer synchronise, must lead to a break of the session between the computers involved. In such a situation it is absolutely necessary for both computers and the line between them to be left in a defined way, thus allowing a new session to start. Because it is possible that both computers could

be shut down during an open session, it is indispensable that both computers control each other and are able to close the session. Here it is still the same situation as described above: most of the exceptional situations can only be detected by using timeout variables.

6. DATA TRANSFER

What kind of data must be transferred between the two servers? On the one hand it is the name of the desired library subroutine and on the other it is the parameter data. Because we already know the names of all executable library routines, we can use a simple coding for the names.

Now to the parameter data. The main axiom for the data transfer in such an environment is that the control and coding overhead has to be reduced. In general the data must be converted from the PC data format to the host data format and vice versa, because we are working in a heterogeneous environment. Thus the designer of such a distributed system must have an excellent knowledge of the data representation on the involved computers. After converting the data we must be sure that we only transfer printable characters, as explained above. How to do this? One can either convert the binary data into hexadecimal presentation, or one can use a three-to-four coding algorithm as described in Ref. 10.

The protocol of the transfer session can be defined as follows: at first a so-called procedure header is transferred. The contents of this header can be the name of the subroutine which should be called, and for example the dimension of variable arrays. This header is followed by data packets of a size of 64 bytes for example. This is a packet size found by experiment with our prototype system. Each packet which is transferred is acknowledged by the host. Finally the result parameters were transferred in the same way. The difference is only that the procedure header can then contain an error message.

7. CONCLUSION

For the professional usage of personal computers in a scientific environment it is necessary to have access to libraries residing on mainframes. Otherwise the personal computers will not be a powerful aid in the field of scientific computation – they will be degraded to a kind of typewriter. Therefore this paper is to be considered as a base for a system designer, showing how to structure a simple distributed system for the remote use of library subroutines. The modularising concepts which were shown for the two communication servers demonstrate that it is possible to divide problems of distributed computing into clearly defined functional units. That these concepts are working well has been proved with the prototype ICARUS.[10]

REFERENCES

1. IMSL, *International Mathematical and Statistical Library, Reference Manual.* IMSL, Houston (1982).
2. J. J. Dongarra, C, B. Moler, J. R. Bunch and G. W. Stewart, *Linpack – Users' Guide.* SIAM, Philadelphia (1979).
3. ISO/TC97/SC16, *Open Systems Interconnection, Basic Reference Model, Draft Proposal,* ISO/DP 7498 (1980).
4. ISO/TC97/SC16, *International Organization for Standardization, Draft Transport Service Specification,* ISO/TC97/SC16 N563 (1980).
5. S. K. Shrivastava and F. Panzieri, *Reliability Aspects of Remote Procedure Calls.* Tutorial paper, University of Newcastle upon Tyne Computing Laboratory (1984).
6. A. D. Birrell and B. J. Nelson, Implementing remote procedure calls. *Transactions on Computer Systems, ACM,* **2**, (1), (1984).
7. B. J. Nelson, *Remote Procedure Call.* Department of Computer Science, Carnegie-Mellon University, CMU-CS-81-119, Pittsburgh, (1981).
8. B. Liskov, *Programming Languages: Issues in Process and Communication Structure for Distributed Programs.* Brown, Boveri & Co. 1985 Symposium on Computer Systems for Control Processes, Baden, Switzerland.
9. N. Wirth, *The Personal Computer Lilith.* ETH-Zürich, Institut für Informatik, Report No. 40, Zürich (1981).
10. B. Teufel, ICARUS – ein System zur nicht-lokalen Prozedurausführung. *Angewandte Informatik* **27** (7) (1985).
11. N. Wirth, *Programming in MODULA-2.* Springer-Verlag, Berlin, (1983).
12. S. M. Shatz, Communications mechanisms for programming distributed systems. *IEEE Computer* **17**, (6) 21–28, (1984).
13. P. Beck *et al., Übertragung von Dateien in einer heterogenen Rechnerumgebung – Entwurf der Arbeitsgruppe für Filetransfer.* ETH-Zürich, Zürich (1982).